8K BASIC VER. 2.3

Written By

Robert H. Uiterwyk
4402 Meadowwood Way
Tampa, Florida 33624

USER'S GUIDE

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 W. RHAPSODY SAN ANTONIO, TEXAS 78216

NOTICE TO USERS OF SWTPC PAPER AND CASSETTE TAPES

In order to help reduce the time necessary to load programs through either a paper tape reader or an SWTPC AC-30 cassette interface, the longer tapes supplied from SWTPC will be furnished in a binary format instead of the conventional ASCII. At the beginning of each tape is a binary loader program that will load into the computer using the regular ASCII format. The program then executes itself and loads the main program in binary. Using this method, tapes will load in approximately 1/3 normal time. When using an SWTPC AC-30, lock the reader in the ON position and type L. For paper tapes readers, such as on an ASR-33 Teletype,® when the load stops after the binary loader has been loaded into the computer simply type G. This will execute the binary loader and the remainder of the tape will load into memory. Several "garbage" characters may be printed immediately after the binary loader loads in—this is normal. On cassette tapes, one side will be in conventional ASCII (side with long leader) and one side will be in binary. The tapes are formatted as follows:

L	BINARY LOADER IN ASCII	S9	G	MAIN PROGRAM IN BINARY

As the tape loads, you will see one of the following displays on your terminal: (either is OK)

*L	*L
*G	*??
* (register dump)	* (register dump)

Some tapes may have an additional feature which will verify that the tape loaded correctly into memory. If, after loading the tape, you find that the program counter is not automatically set to the correct value then you probably have a verifying tape. If this is the case simply typing a G will automatically check the validity of the program and execute it. If the message LOAD ERROR is displayed then the tape did not load correctly into memory. The most common cause of this is a memory problem—there can be problems that MEMCON and ROBIT will not find.

The format for a self-verifying tape is as follows:

VERIFICATION ROUTINE	BINARY LOADER PGM. CTR.	BINARY LOADER IN ASCII	S9	G	MAIN PROGRAM IN BINARY	MAIN PROGRAM PGM. CTR.
ASCII					BINARY	

As before, one side of a cassette tape will be in binary and the other side in ASCII.

If you are unable to load a tape please check the following:

1.) Be sure the reader is locked on to load a binary cassette tape.
2.) Try different volume and tone control settings.
3.) Clean your tape heads with alcohol and a cotton swab.
4.) Re-check all memory if a LOAD ERROR is displayed.

SWTPC 8K BASIC Ver. 2.3

SWTPC 8K BASIC is a complete BASIC interpreter for use in both home and personal applications. Features of SWTPC 8K BASIC include nine significant digit binary coded decimal addition, subtraction, multiplication and division, seven digit trigonometric functions and numerous string operations.

This manual is designed to acquaint the user with the various features of SWTPC BASIC—It is not designed to be a complete course on the BASIC language.

Definitions

Before actually describing each BASIC function, several terms need to be defined and manual notation described.

A **command** is a BASIC operation that generally has an immediate effect on the operation of BASIC.

A **statement** is a word or group of words that directs the execution of a BASIC program.

A **function** is a BASIC operation that usually results in a numerical operation or string processing.

A **variable** is a letter, or a letter and a number, that is used to represent a numeric or string value. Variables may be named by any single alphabetic character (A-Z) or any single alphabetic character followed by a number (0-9). Variables of this type represents a numerical value.

Example: A can be equated to 3.44
B1 can be equated to -7.2315 + SIN(3)

A **string variable** is a single letter followed by a $ that is used to represent literal (alphanumeric or text) data.

Example: A$ can be equated to "1234" but not to 1234. (the quotation marks make 1234 a string).
Note: a string may **not** be represented by a letter and a number such as A3$.

When BASIC initializes, the string variable length is set equal to a maximum of 32 characters.

This manual uses the following notation conventions:

/line N/	denotes a BASIC line number such as 0090
/var/	denotes a variable name such as A3
/exp/	denotes a mathematical expression such as 3+5-2
/rel.exp./	denotes a relational expression such as A=5
/string/	denotes a collection of literal alphanumeric characters enclosed by quotation marks such as "TEST1"
X	denotes a variable or expression that has a numerical result
X$	denotes a string variable

Restrictions on Program Lines

The following restrictions are placed on all BASIC program lines:

1.) Every line must have a line number ranging between 1 and 9999. Do not use line # 0.
2.) Line numbers are used by BASIC to order program statements sequentially.
3.) In any program, a line number can be used only once.
4.) A previously entered line may be changed by entering the same line number along with the desired statement(s). Typing a line number followed immediately by a carriage return deletes that line and line number.
5.) Lines need not be entered in numerical order since BASIC will automatically order them in ascending order.
6.) A line may contain no more than 72 characters including blanks.

7.) Blanks, unless within a character string and enclosed by quotation marks, are not processed by BASIC and their use is optional. Numbers can contain no imbedded blanks.
Example:

```
110 LET A=B + (3.5*5E2)
    is equivalent to
110 LETA = B+(3.5*5E2)
```

8.) Multiple statement lines are accepted with a colon (:) used as the separator. BASIC will process the line from left to right.
Example:

```
10 A=3 : B=5 : C=A*B
```

Data Format

The range of numbers that can be represented in this version of BASIC is 1.0E-99 to 9.999999-99E99. E99 represents 10^{99} while E-99 represents 10^{-99}. The E stands for exponent.

There are nine digits of significance in this version of BASIC. Numbers are internally truncated (last digits dropped) to fit this precision.

Numbers may be displayed and entered in three formats: integer, decimal and exponential.

Example: 153 34.52 136 E-2

Transcendental functions (SIN, COS, TAN, ATAN, SQR, LOG and ↑) are all evaluated by a limited infinite series. For these functions accuracy is limited to seven significant digits.

Mathematical Operators

The mathematical operators used in BASIC are as follows:

↑	Exponentiation (raises to a power)
– (unary)	Negate (used for denoting negative numbers
+	Addition and string concatenation
-	Subtraction
*	Multiplication
/	Division

No two mathematical operators may appear in sequence, and no operator is ever assumed. (A++B and (A+2)(B-3) are not valid). Exception: 5+-3 is allowed.

Examples:

A=B↑C	A is evaluated to B raised to the C power
A=B+-5	A is evaluated to B plus a negative 5
A=3/2	A is evaluated to 3 divided by 2

Priority of Operations

BASIC recognizes the priority of operation in the following order:

1. Exponentiation (↑)
2. Negation (-)
3. Multiplication (*) and division (/)
4. Addition (+) and subtraction (-)

A BASIC expression is evaluated from left to right in the above priority sequence unless parenthesis are encountered. The operators within the parenthesis are evaluated first utilizing the above priority structure.

Examples:

```
LET A=2
LET B=3
LEC C=4
B↑2 + C/A↑2      gives a result of 10
C↑2 - C/A        gives a result of 14
A * (A+B*2)-22   gives a result of 0
A↑A↑B            gives a result of 64
```

String Concatenation

Although any one string variable may be a maximum of 32 characters (or whatever the length is set equal to using the STRING= command), strings may be joined up to a maximum of 128 characters for printing. The concatenation symbol is +.

```
Example: A$= (32 char. string)
         B$= (32 char. string)
         PRINT A$+B$ (prints a 64 character string)
   also: A$= "HELLO"
         B$= "JOHN"
         C$= A$ + B$    (C$ still limited to 32 char.)
```

Arrays

Sometimes it is convenient for a variable to represent several values at one time. A variable such as this type can be considered as an array and each element can be accessed independently. In referencing an array variable, the element number in the array must be specified along with the variable name. For example, say we wanted the variable A to represent 4 values. The following program would assign a different value to each element of A.

```
10  DIM A(4)      Dimension A to hold four elements
20  A(1)=1 : A(2)=2 : A(3)=3 : A(4)=4
```

As seen above, a particular element is referenced by a subscript N, such as A(N), where 1 is the first element in the array.

Two dimensional arrays are also accepted by BASIC. Two dimensional arrays are useful when working with data which is easily represented as a matrix.

```
Example:    10  DIM A (3,3)
            20  A(1,1)=1 : A(1,2)=2 : A(1,3)=3
            30  A(2,1)=4 : A(2,2)=5 : A(2,3)=6
            40  A(3,1)=7 : A(3,2)=8 : A(3,3)=9
gives the following matrix:    1  2  3
                               4  5  6
                               7  8  9
```

String variables may also be dimensioned as arrays. (A$(5,2))

If no DIM statement is used to specifically dimension an array, a dimension of either 10 or 10 by 10 is assumed.

Program Preparation and System Operation

At the time that BASIC is executed, BASIC will automatically determine the range of working storage. If you wish to limit the amount of memory BASIC uses, refer to Appendix C of this manual. This is normally not necessary unless external machine language subroutines are being used.

The system is then ready to accept commands or lines of statements. For example the user might enter the following program:

```
150  REM DEMONSTRATION
160  PRINT "ENTER A NUMBER";
170  INPUT A
180  LET P = A*A*3.1415926
185  PRINT
190  PRINT "THE AREA OF A CIRCLE WITH";
200  PRINT "RADIUS"; A; "IS"; P
210  STOP
```

If the user wishes to insert a statement between two others, he need only type a statement number that falls between the other two. For example:

```
183  REM THIS IS INSERTED BETWEEN 180 and 185.
```

If it is desired to replace a statement, a new statement is typed that has the same number as the one to be replaced. For example:

180 P=(A*A)*3.1415926 replaces the previous LET statement.

Each line entered is terminated by a Carriage Return and is not processed by BASIC until this key is depressed. BASIC then positions the print unit to the correct position on the next line.

If a mistake is made during type in before typing the Carriage Return, a BACKSPACE may be used to delete erroneous characters. The backspace character for BASIC is a hexadecimal ASCII Ø8 (Control H). BASIC assumes that the terminal automatically generates a "cursor left" when a control H is entered.

Example:

30 REM THIS IS A TESZ (CTL.H)T

The CTL.H moves the cursor back over the Z so that the result is TEST

If it is desired to remove a complete line that was typed in before typing the Carriage Return, the CANCEL key (hex ASCII 18, control X) may be depressed. This will delete all information that was typed in on the current command or statement line. BASIC will respond with DELETED.

Example:

10 FOR 1 to 10 (CTL.X)
DELETED

PATCH (CTL. X)
DELETED

If the user wishes to execute a program at this point, the RUN command , as described in the command section, should be entered.

Program Abort

If, at any time, it is desired to abort a looping or otherwise malfunctioning program, BASIC has a provision for exiting the program and returning to the command (READY) mode. The abort (break) character for BASIC is a control C , hex ASCII 03. The actual operation of the control C varies somewhat depending on the type of interface used on the control port.

MP-C Control Interface

This type of interface requires hitting the control C key very rapidly a number of times for aborting a program. It is sometimes normal for several question marks or extraneous characters to be displayed while hitting control C. BASIC should then respond READY.

MP-S Serial Interface (6800/2 Owners)

When using this type of interface, entering one control C will immediately halt the execution of the current BASIC program and will return BASIC to the command mode. During a printout sequence, such as listing a program, typing one ESC (escape) character will cause the current printout to halt. Typing another ESC will cause printout to resume while typing a CONTROL C will force BASIC back to the command mode.

NOTE:When in the middle of a machine code USER routine, control C will have no effect. If necessary, the computer's RESET button can be depressed. Resetting the computer's program counter to 0103 before re-entering BASIC will keep the current BASIC program intact.

Commands

It is possible to communicate in BASIC by typing direct commands at the terminal device. Also, certain other statements can be directly executed when they are entered without statement numbers.

Commands have the effect of causing BASIC to take immediate action. A typical BASIC language program, by contrast, is first entered statement by statement into the memory and then executed only when the RUN command is given.

When BASIC is ready to receive commands, the word READY is displayed on the terminal device. After each entry, the system will prompt with a "#".

Commands are typed without statement numbers. After a command has been executed, READY will again be displayed indicating that BASIC is ready for more input—either another command or program statements.

APPEND

The APPEND command causes a program on tape to be loaded into memory. The APPEND command operates the same as the LOAD command except that the current BASIC program is not cleared from memory.

CONT

A CONT (continue) command can be entered after a program has halted from a STOP command or after a program has been aborted with a control C. Between the time that the program has stopped and the time that CONT is entered no changes should be made in the program. The program will then continue with the next statement after the STOP command or wherever the program was when control C was pressed.

DIGITS=X

The DIGITS= command sets the number of digits that will be printed to the right of the decimal point when displaying numeric variables. This will truncate (not round) any digits greater than the number printed, and will force "0"s if there aren't enough significant digits to fill the number of positions specified in the "DIGITS =" command.

A DIGITS=0 command resets BASIC to the floating point mode.

The DIGITS= command may also be used as a program statement.

LINE=X

The LINE= command is used to specify the number of print positions in a line (line length) where X is the desired number of print positions.

Example: LINE=65, LINE=80, LINE=40

Note: Each line is broken up into 16 character "zones". If the print position is within the last 25 percent of the "line" length and a "space" is printed, a C/R L/F will be output. This is so that a number or word will **not** be split up at the end of a print line. If it is desired to inhibit this feature (for precise print control) just set the line length equal to greater than 125% of the desired total print line length. This can be very important when using the TAB command.

The LINE= command can also be used as a program statement.

LIST
LIST (line #)
LIST (line #m)-(line #n)

The LIST command causes the desired lines of the current program to be displayed on the control terminal. The lines are listed in increasing numerical order by line number. A LIST command causes all lines of the current program to be displayed, a LIST (line #) command lists only the line specified and a LIST (line #m)-(line #n) command causes all lines from m to n to be listed.

The LIST command can also be used to output lines to a terminal or printer on another port by entering #N, after LIST (such as LIST #7, 110-130) where N is the desired port number.

Examples: LIST
LIST 30
LIST #3, 30-70

The LIST command can also be used as a program statement.

LOAD

The LOAD command is used for loading BASIC programs previously saved on cassette and paper tapes. All input/output regarding the LOAD command will be thru the control or defined port. Appropriate reader on/off commands are automatically generated.

If desired, the input from LOAD can be channeled thru a port other than the control port by using the LOAD #N command where N is the desired port number. The same rules apply for port types and handshaking as described in the PORT=command.

NOTE: Both the LOAD and SAVE commands assume that the punch/read device is set up to decode automatic reader/punch on/off commands. If your particular unit is not automatic (such as an AC-30 on port Ø), the reader or punch should be turned on manually before the carriage return is entered after typing the respective SAVE or LOAD command. See Appendix G for more information on using port #Ø as a SAVE/LOAD channel.

NEW

The NEW command causes the working storage and all variables and pointers to be reset. The effect of this command is to erase all traces of the previous program from memory. This command also sets LINE equal to 48 and DIGITS equal to Ø (floating point mode).

PATCH

The PATCH command causes computer control to be returned to the computer's monitor.

PORT=X

The PORT=X command defines the computer I/O Port which will serve as the 'Control Port'. "X" can be a constant, variable, or expression.

Example: PORT = 3

Warning—If you define a port without a terminal as the control port, all messages (including the "Ready") will be inputed and outputed from that port. . .therefore, you will lose control of your program.

NOTE: PIA ports **require** handshaking. If handshaking is not available, then you must use the PEEK command to examine the PIA registers. Also, BASIC will always accept a break from port 1, therefore never leave port 1 without a terminal connected. Appendix F defines the correct handshaking procedure. Each port # is configured by BASIC for the specified type of interface:

PORT	TYPE OF PORT
Ø	MODIFIED PIA (MP-C interface)
1	MODIFIED PIA (CONTROL PORT) or ACIA
2	ACIA
3	ACIA
4	PIA
5	PIA
6	PIA
7	PIA (LINE PRINTER, BY CONVENTION, such as SWTPC PR-40)

The PORT command can also be used as a program statement.

RUN

The RUN command causes the current program resident in memory to begin execution at the first statement number. RUN always begins at the lowest statement number. RUN resets all program parameters and initializes all variables to zero.

SAVE

The SAVE command is used for saving BASIC programs onto cassette or paper tape. All output from the SAVE command will be thru the control or designated port. Appropriate punch on/off commands are automatically generated for use by the tape storage device.

If desired, the output from SAVE can be directed to another port by using the SAVE #N command where N is the desired port number. The same rules apply for port types and handshaking as described in the PORT=command.

STRING=X

The STRING= command sets the maximum allowable length of string variables. The STRING= command may be used as part of a program and must be used before any strings are referenced in a program. X may be any number between 1 and 128. STRING is initially set to 32 characters. The NEW command will not reset the string length to 32.

TRACE ON

The TRACE ON command will cause BASIC to display the line number of the current statement being executed for every line. This can be an important debugging tool.

TRACE OFF

The TRACE OFF command turns off the trace function.

A statement, in BASIC, is a word or a group of words that directs the execution of a BASIC program. Statements differ from commands in that they generally do not cause the computer to immediately take action by themselves. Some statements, in fact, must be used with other statements for proper operation.

DATA N1, N2, N3, . . .
READ V1, V2, V3, . . .
RESTORE

The DATA, READ and RESTORE statements are used in conjunction with each other as one of the methods to assign values to variables. Every time a DATA statement is encountered, the values in the argument field are assigned sequentially to the next available position of a data buffer. All DATA statements, no matter where they occur in a program, cause DATA to be combined into one list.

READ statements cause values in the data buffer to be accessed sequentially and assigned to the variables named in the READ statement. They start with the first data element from the first data statement, then the second element, to the end of the first data statement, then to the first element of the second data statement, etc., each time a READ command is encountered. If a READ is executed, and the DATA statements are out of data, an error is generated.

Numeric and string data may be intermixed, however it must be called in the appropriate order.

Note: String data need not be enclosed within quotes (") as the comma (,) acts as the delimiter. However, if the string contains a (,), then it must be delimited by quotes (").

Example:

```
10 DATA 10,20,30,56.7,"TEST,ONE",1.67E30,8, HELLO
20 READ A,B,C,D,E$,F,G5,F$
```

Note: DATA STATEMENTS may be placed anywhere within the program.

Example:

```
110 DATA 1,2,3.5
120 DATA 4.5,7,70
130 DATA 80,81
140 READ B,C,D,E
```

is the equivalent of:

```
10 LET B=1
20 LET C=2
30 LET D=3.5
40 LET E=4.5
```

The RESTORE statement causes the data buffer pointer, which is advanced by the execution of READ statements, to be reset to point to the first position in the data buffer.

Example:

```
110 DATA 1,2,3.5
120 DATA 4.5,7,70
130 DATA 80,81
140 READ B,C
150 RESTORE
160 READ D,E
```

In this example, the variables would be assigned values equal to:

```
100 LET B=1
101 LET C=2
102 LET D=1
103 LET E=2
```

There are also versions of READ and RESTORE which are used for the manipulation of disk data files. These statements are discussed in the Disk Data Files section.

DIM/var/ (exp) or /var/ (exp), /var/(exp) or /var/(exp,exp)

The DIM statement allocates memory space for an array. In this version of BASIC, one or two dimension arrays are allowed and the maximum array size is 255 x 255 elements. All array elements are set to zero by the DIM statement.

If an array is not explicitly defined by a DIM statement, it is assumed to be defined as an array of 10 elements (or 10 x 10 if two elements are used) upon first reference to it in a program.

Caution: The dimension of an array can be detemined only once in a program, implicitly and explicitly. Also only the variables A thru Z (followed by $) may be dimensioned for strings.

Example: DIM A(10), C(R5+8), D(30,A*3), A7(20), C$(30), Z$(5)
but not A6$(5)

The DIM statement can also be used in the direct execution mode.

END

The END statement causes the current BASIC program to stop executing. When an END statement is seen, BASIC will return to the command mode. In this version of BASIC, END may appear more than once and need not appear at all.

FOR /var/ = /exp 1/ TO /exp 2/ STEP /exp/
NEXT /var/

The FOR and NEXT statements are used together for setting up program loops. A loop causes the execution of one or more statements for a specified number of times. The variable in the FOR. . .TO statement is initially set to the value of the first expression (exp1). Subsequently the statements following the FOR are executed. When the NEXT statement is encountered, the values of the named variable is added to the value specified by the STEP expression in the FOR. . . TO statement, and execution is resumed at the statement following the FOR. . .TO. If the addition of the STEP value develops a sum that is **greater than** the TO expression (exp2) or, if STEP is negative, a sum **less than** the TO expression (exp2), the next instruction executed will be the one following the NEXT statement. If no STEP is specified, a value of one is assumed. If the TO value is initially less than the initial value, the FOR. . .NEXT loop will still be executed once.

```
Example:  110 FOR I=.5 TO 10
          120 INPUT X
          130 PRINT I,X,X/5.6
          140 NEXT I
```

Although expressions are permitted for the initial, final, and STEP values in the FOR statement, they will be evaluated only the first time the loop is entered. They are not re-evaluated.

It is not possible to use the same indexed variable in two loops if they are nested.

When the statement after the NEXT statement is executed, the variable is equal to the value that caused the loop to terminate, not the TO value itself. In the above example, I would be equal to 9.5 when the loop terminates.

GOSUB /line #/

A subroutine is a sequence of instructions which perform some task that would have utility in more than one place in a BASIC program. To use such a sequence in more than one place, BASIC provides subroutines and functions.

A subroutine is a program unit that receives control upon execution of a GOSUB statement. Upon completion of the subroutine, control is returned to the statement following the GOSUB by execution of a RETURN statement in the subroutine.

```
Example:  10 A=3
          20 GOSUB 100
          30 PRINT B
          40 END
         100 LET B= SIN(A)
         110 RETURN
```

GOTO /line #/

The GOTO statement directs BASIC to execute the statements on the specified line unconditionally. Program flow continues from the line specified by /line/.

Example: 150 GOTO 270

This statement may be used in the direct execution mode.

IF /relational exp/ THEN /statement n/
IF /relational exp/ THEN /BASIC statement/ (Direct)

The IF statement is used to control the sequence of program statements to be executed, depending on specific conditions. If the /relational expression/ given in the IF is "true", then control is given to the line number declared after the THEN. If the relational expression is "false", program execution continues at the line following the IF statement.

Example: 10 IF 5>2 THEN 100

It is also possible to provide a BASIC statement after the THEN in the IF statement. If this is done and the relational expression is true, the BASIC statement will be executed and the program will continue at the statement or line following the IF statement.

Example: 10 IF 5>2 THEN LET B=7

When evaluating relational expressions, arithmetic operations take precedence in their usual order, and the relational operators are given equal weight and are evaluated last.

Example: 5+6*5>15*2 evaluates to be true

The Relational Operators

=	Equal
<>	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal
>=	Greater Than or Equal

Examples:
```
110 IF A<B+3 THEN 160
180 IF A=B+3 THEN PRINT "VALUE A", A
190 IF A=B THEN T1=B
```

NOTE: If an IF test fails on a multiple statement line, the remainder of the line will not be executed.

Example:
```
10 IF 5<2 THEN 100 : PRINT 3
20 END
```
Control will go to line 20 and "3" will not be printed

The relational operators = (equal) and <> (not equal) may also be used on strings.

Example: 110 IF Y$ = "YES" THEN 100

The < (less than) and > (greater than) comparisons may also be used on strings, but **only** when the number of characters in each of the strings being compared is the same. The > and < operators compare strings by evaluating the ASCII value of the characters starting from the first (leftmost) character. When a character in one string is found to be not equal to its respective character in the other string, the greater than or less than operation is made either true or false depending on the ASCII values of these two characters.

Example: IF "AAABA" > "AAAAB" THEN 100

The first non-equal character in the comparison is the B in "AAABA". The > operator then compares this B to the respective character in the other string (an A). Since the ASCII value of B is greater than that of A, the operation evaluates to "yes, greater than".

Example:

"A" > "B"	FALSE
"B" > "A"	TRUE
"ABCDE"<"ABCDF"	TRUE
"ABC" > "ABCD"	ILLEGAL, LENGTHS NOT EQUAL
"BZZ" > "CZZ"	FALSE

INPUT /var/
INPUT /var/, /var/, /var/,. . .
INPUT #N, var
INPUT "/PROMPT/",/var/

The INPUT statement allows users to enter data from the terminal during program execution.

Example:
INPUT X - Inputs one numeric value
INPUT X$ - Inputs one string value
INPUT X,Y,Z,B$ - Multiple inputs may be entered, separated by ",". If the expected number of values are not entered, another "?" will be generated.
INPUT "ENTER VALUE",X - Prints the message in quotes, then a "?", and waits for input. It stores the inputed value in X.

When the program comes to an INPUT statement, a question mark is displayed on the terminal device. The user then types in the requested data separated by commas and followed by a carriage return. If insufficient data is given, the system prompts the user with '?'. If no data is entered, or if a non-numeric character is entered, the system prompts "RE-ENTER". However, for string variables a null return will be considered as valid data. Caution: for input A$,B$,C$—a null response would create three null variables. Only constants can be given in response to an INPUT statement.

The INPUT can also be used to issue a prompting message before the question mark appears.

Example:
```
10 INPUT "INPUT A$", A$
20 PRINT A$
would give the following results
INPUT A$ ? 66  (user types this 66 in)
66
```

INPUT may also be used with the #N, directive for input from ports other than the control port.

LET /var/=/exp/

The LET statement is used to assign a value to a variable. The use of the word LET is optional unless you are in the direct mode.

Example:
```
100 LET B=827
110 LET C=87E2
120 R=(X*Y)/2*A
130 C$="THIS IS TEXT"
```

The equal sign does not mean equivalence as in ordinary mathematics. It is the replacement operator. It says: replace the value of the variable named on the left with the value of the expression on the right. The expression on the right can be a simple numerical value or an expression composed of numerical values, variables, mathematical operators, and functions.

ON /exp/ GOTO /line (s)/
ON /exp/ GOSUB /line(s)/

This statement transfers control to the line or subroutine as defined by the value of /exp/. The expression will be evaluated, truncated (chopped off after the decimal point) and control then transferred to the nth statement number (where n is the integer value of the expression).

Example: ON N GOTO 110, 300, 500, 900
Means:
```
If N <1You will get an error
If N=1 GOTO 110
If N=2 GOTO 300
If N=3 GOTO 500
If N=4 GOTO 900
If N>4 You will get an error
```
Example: ON (N+7)*2 GOSUB 1000,2000

(see GOTO and GOSUB for a further description of these statements)

PRINT /var/
PRINT /string/
PRINT /exp/

The PRINT statement directs BASIC to print the value of an expression, a literal value, a simple variable, or a text string on the user's terminal device. The various forms may be combined in the print list by separating them with commas or semicolons. Commas will give zone spacing of print elements, while semicolons will give a single space between elements. If the list is terminated with a semicolon, the line feed/carriage return at the end will be suppressed.

1. PRINT – Skips a line.
2. PRINT A,B,C – Prints the values of A, B, and C, separated into 16 space zones. Use of a ";" in place of the "," would print A, B, and C separated by one space. (No space is generated if a string variable.) A C/R, L/F is generated at the end of the line.
3. PRINT "LITERAL STRING" – Prints the characters contained within the quotes.
4. PRINT A,B;"LITERAL"–Prints variable A & B and the word LITERAL.

PRINT may also be used with the #N directive to specify output to another port.

Example:
```
10 PRINT #7, "TEST"
```
Prints TEST on the parallel device (printer, etc.) on port #7.

PRINT may also be used in the direct mode.

REM

The REM, or remark statement, is a non-executable statement which has been provided for the purpose of making program listings more readable. By generous use of REM statements, a complex program may be more easily understood. REM statements are merely reproduced on the program listing, they are not executed. If control is given to a REM statement, it will perform no operation. (It does, however take a finite amount of time to process the REM statement.)

Example:
```
120 REM THE FOLLOWING SUB. CONVERTS
121 REM DECIMAL VALUES TO HEX VALUES
```

RETURN

The GOSUB statement causes control to be passed to the given line number. It is assumed that the given line number is an entry point of a subroutine. The subroutine returns control to the statement following the GOSUB statement with a RETURN statement.

Example:
```
100 X=1
110 GOSUB 200
120 PRINT X
125 X=5.1
130 GOSUB 200
140 PRINT X
150 STOP
200 X=(X+3)*5.32E3
210 RETURN
211 END
```

Subroutines may be nested; that is one subroutine may use GOSUB to call another subroutine which in turn can call another. Subroutine nesting is limited to eight levels.

STOP

A STOP statement can be used within a program to halt execution at a particular place for debugging purposes. A CONT command will then cause the computer to begin execution on the line following the STOP statement.

Example:
```
10 PRINT 5
20 STOP
30 PRINT 6
```
gives the following output:
```
RUN
```

```
5
STOP AT 20
CONT
6
```

FUNCTIONS

Functions are similar to BASIC statements except that they usually relate to mathematical or string processing operations.

ABS (X)

The ABS (X) function returns the "Absolute Value" of X.

Example: ABS (3.44)=3.44
ABS (-3.44)=3.44

ATAN (X)

The ATAN (X) function returns the angle, in radians, whose tangent is X.

ASC (string or string var)

The ASC (string or string variable) function returns the **decimal** ASCII numeric value of the **first** ASCII character within the string. Literals must be enclosed by quotes while string variables are not.

Example:
```
ASC("?")          gives 63
ASC("A")          gives 65
ASC("B")          gives 66
ASC("Z")          gives 90
ASC("5")          gives 53
LET B$="5" -> >ASC(B$)     gives 53
```

CHR$ (X)

The CHR$ (X) function returns a single character string equivalent to the decimal ASCII numeric value of X. This is the **inverse** of the ASC function.

Example:
```
CHR$(63)    gives a ?
CHR$(65)    gives a A
CHR$(66)    gives a B
CHR$(53)    gives a 5
```

COS(X)

The COS(X) function returns the cosine of the angle X. X must be in radians.

DEF FN/letter/(/variable/)=/exp/

This function allows the user to create his very own functions. The /letter/ is any alphabetic character. This names the function (i.e., you could have, say, three functions named FNA, FNB, and FNC). The /variable/ is a non-subscripted numeric variable. This is essentially a "dummy" variable (or place holder). . .This will be apparent shortly. The "Expression" is any valid expression. Note that the "variable" **must** be enclosed within parenthesis.

For example, study the following sample program:

```
10 DEF FNA(X)=3.14*X↑ 2
20 DATA 5,6,7,0
30 READ X
40 IF X=0 THEN END
50 PRINT FNA(X)
60 GOTO 30

RUN
```

```
78.5
113.04
153.86

READY
```

As you can see, the dummy variables were replaced with the variables you actually wished to use at the time the function was used.

Note: You may **not** define the same function greater than once per program, and a function must be defined before it is called.

EXP(X)

The EXP(X) function returns the base of natural logarithms raised to the Xth power (this is the inverse of LOG(X)) and is the equivalent of 2.718282 raised to the Xth power.

INT(X)

The INT(X) function returns the greatest integer **less than** X.

Example: INT(4.354)=4

Now note this one: INT(-4.354)=-5

LEFT$(X$,N)

The LEFT$(X$,N) function returns a string of characters from the leftmost to the Nth character in X$.

```
Example:    X$="ONE,TWO,THREE"
            LET A$=LEFT$(X$,6)
            A$ NOW EQUALS "ONE,TW"
```

LEN(X$)

The LEN(X$) function returns the number of characters contained in string X$.

```
Example:    LEN("TESTING")=7
            LEN("TEST ONE")=8
```

Note: The space **does** count.

Hint: LEN(STR$(X)) = The number of print positions required to print the number X.

LOG(X)

The LOG(X) function returns the natural logarithm of the number X.

MID$(X$,X,Y)

The MID$(X$,X,Y) function returns a string of characters from X$ beginning with the Xth character from the left, and continuing for Y characters from that point. Y is **optional.** If Y is **not** specified, then the string returned is from the Xth character of the string through the end of the string.

```
Example:    X$="ONE,TWO,THREE"
            A$=MID$(X$,3,10)
            A$ NOW EQUALS "E,TWO,THRE"
```

PEEK(X)

The PEEK(X) function returns, in **decimal,** the value contained in **decimal,** not octal, memory location X.

```
Example:    LET A=PEEK(255)
```

A will now contain the decimal value contained in memory location 255_{10}.

POKE(I,J)

The POKE(I,J) function takes the decimal, not octal, value of J and places it in decimal, not octal, location I. For example, POKE (255, 10) will store a decimal 10 in decimal memory location 255.

Warning: This function can cause system program failure if improperly used.

POS

The POS function returns in decimal, not octal, the current position of the print head or cursor. The first postion (left margin) is position #1.

RIGHT$(X$,N)

The RIGHT$(X$,N) function returns a string of characters from the Nth position to the left of the rightmost character through the rightmost character.

```
Example:    X$="ONE,TWO,THREE"
            A$=RIGHT$(X$,9)
            AS NOW EQUALS "TWO,THREE"
```

RND AND RND(X)

The RND(X) function produces a set of uniformly distributed pseudo-random numbers. If X (the seed) is Ø, then each time RND(X) is accessed, a different number between Ø and 1 will be returned. If X <> Ø then a specific random number will be returned each time (the same number each time). RND can be called without an argument, in which case it works as if one had used an argument of Ø.

```
Example:    10 LET A=RND
            20 LED B=RND(5)
```

If you require random numbers other than between 0 and 1, then:

```
            PRINT INT ( (B-A+1)*RND(0)+A)
```

will yield random numbers ranging between A & B.

SGN(X)

The SGN(X) function returns the 'sign' (+,-, or 0) of X. The SGN of a negative number will yield a -1, the SGN of a positive number will yield 1 and the SGN of 0 gives 0.

```
Example:    SGN(4.5)=1
            SGN(-4.5)=-1
            SGN(0)=0
            SGN(-0)=0
```

SIN(X)

The SIN(X) function returns the sine of the angle X. X must be in radians.

SQR(X)

The SQR(X) function returns the square root of X. X must be greater than or equal to 0 (X must be positive).

STR$(X)

The STR$(X) function returns the string value of a numeric value. This is the inverse of the VAL function.

```
Example:    A=34567
            LET A$=STR$(A)
            A$ NOW EQUALS "34567"
```

TAB(X)

The TAB(X) function will move the print position to the "Xth" position to the right of the left margin. If the print position is already to the right of the position specified in the TAB command, no spaces will be left and printing (if any) will commence. The first print position (left margin) is position #1.

The TAB function can be used with the PRINT statement to cause data to be printed in exact locations. The argument of TAB may be an expression.

Example:

```
  5   X=3
 10   PRINT TAB(2); X; TAB( );X*X; TAB( ); X*X*S
will print
  3        9        27
```

TAN(X)

The TAN(X) function returns the tangent of the angle X. X must be in radians. (360 degrees= 2π radians π =3.141592654

USER(X)

The USER (X) function is a BASIC function that enables a user to call a special machine language subroutine. The syntax of the USER function is of the form LET /var/=USER (/var.1/) such as LET A = USER(X). The use of the USER function assumes that the programmer is familiar with assembler level programming.

When the USER function is executed in a program, the numeric value of the variable X is stored in a special BCD (binary coded decimal) format in a seven byte series somewhere in the computer's memory. BASIC then keeps track of where this series is stored so that the machine language routine can access it. After storing this series, BASIC then looks at hex memory locations 0067 and 0068. The computer is then instructed to execute a "Jump to Subroutine" to the hex address stored in hex memory locations 0067 and 0068. To avoid accidental misuse of the USER function, 0067 and 0068 will initialize to a location which contains a hex 39, a return from subroutine. Locations 0067 and 0068 can be changed using the POKE function prior to using USER.

After the computer jumps to the location pointed to by 0067 and 0068 it is up to the machine language program to perform its special function or to manipulate the data previously stored in the seven byte BCD series. To find out where this series is located, hex memory locations 005D and 005E should be checked by the machine program. 005D and 005E contain a pointer to the location of the seven byte series. They do not contain the actual location of the series.

For example, say that locations 005D, 005E contain the address IDB1 This means that locations IDB1 and IDB2 contain the address of the seven byte series. If the series was stored beginning at 242B, then the locations would be set up as follows:

```
005D    1D
005E    B1

IDB1    24
IDB2    2B

242B    Start of seven byte series.
```

The actual number that was stored in the seven byte series is stored in a special BCD format as follows:

for + numbers			for - numbers	
BYTE 1	(sign)	(D_9)	(sign)	($\overline{D}_9$)
BYTE 2	(D_8)	(D_7)	($\overline{D}_8$)	($\overline{D}_7$)
BYTE 3	(D_6)	(D_5)	($\overline{D}_6$)	($\overline{D}_5$)
BYTE 4	(D_4)	(D_3)	($\overline{D}_4$)	($\overline{D}_3$)
BYTE 5	(D_2)	(D_1)	($\overline{D}_2$)	($\overline{D}_1$)
BYTE 6	(Exponent in hex)		(Exponent in hex)	
BYTE 7	00		00	

Where D's are digits and $\overline{D}$'s are the digits complemented.

The sign half-byte denotes whether or not the number is positive or negative. A sign of 0 denotes + while a 9 denotes -. The actual number digits are located in half-bytes D_1 - D_9. The exponent byte tells BASIC where to put the decimal point. Notice that this number is hexadecimal and not BCD.

For example, the number 1234.5678 would be stored as follows:

Byte 1 01
Byte 2 23
Byte 3 45
Byte 4 67
Byte 5 80
Byte 6 04
Byte 7 00

The number is stored as .12345678 with an exponent of 4 which moves the decimal point 4 places to the right giving 1234.5678. The 0 half-byte in byte 1 denotes a positive number.

Now look at the number -1234.5678. Negative numbers are more complicated and must be handled with great care.

Byte 1 98
Byte 2 76
Byte 3 54
Byte 4 32
Byte 5 20
Byte 6 04
Byte 7 00

Notice that the first 9 in byte 1 denotes a negative number and that all digits D_1 - D_9 are complemented. The complement of a digit is defined a 9- (the digit) with the complement of 0 still being 0. In the above example, the digits that were stored were not 12345678 but rather (9-1) (9-2) (9-3) (9-4) (9-5) (9-6) (9-7) and (1+9-8). The **last significant digit not** including any trailing 0's must have 1 added to its complement before storing in the BCD series. In the example -1234.5678 (the same as -1234.56780) the last **significant** digit is 8; therefore, 1 must be added to its complement.

The number -7.20008000 would be stored as:

Byte 1 92
Byte 2 79
Byte 3 99
Byte 4 20 (the last significant digit is 8)
Byte 5 00
Byte 6 01
Byte 7 00

The end of the machine language program should contain a hex 39, a RTS. This will transfer control back to BASIC. BASIC will then assign the numeric value of the number in the seven byte string to the variable A in the example A=USER(X).

VAL(X$)

The VAL(X$) function returns the numeric constant equivalent to the value in X$. This is the **inverse** of the STR$ function.

Example: VAL("12.3")=12.3
VAL("5E4")=5000
VAL("TWO")=GENERATES AN ERROR. "TWO" cannot be equaled to a numeric constant.
VAL("-12.3")=-12.3

APPENDIX A

ERROR MESSAGES

If, during the operation of BASIC, a mistake was made by the programmer, BASIC will output one of the following error messages:

ERROR #	DEFINITION
1.	Oversize variable (over 255) in TAB, CHR, subscript or "ON"
2.	Input error
3.	Illegal character or variable
4.	No ending " in print literal
5.	Dimensioning error
6.	Illegal arithmetic
7.	Line number not found
8.	Divide by zero attempted
9.	Excessive subroutine nesting (max is 8)
10	**RETURN** W/O prior **GOSUB**
11.	Illegal variable
12.	Unrecognizable statement
13.	Parenthesis error
14.	Memory full
15.	Subscript error
16.	Excessive FOR loops active (max is 8)
17.	**NEXT** "X" W/O **FOR** Loop defining "X"
18.	Misnested FOR-NEXT
19.	READ statement error
20.	Error in ON statement
21.	Input Overflow (more than 72 characters on INput line)
22.	Syntax error in DEF statement
23.	Syntax error in FN error, or FN called on Function not defined
24.	Error in STRING Usage, or mixing of numeric and string variables
25.	String Buffer Overflow, or String Extract (in MID$,LEFT$, or RIGHT$) too long
26.	I/O operation requested to a port that does not have an I/O card installed.
27.	VAL function error—string starts with a non-numeric value.
28.	LOG error—an attempt was made to determine the log of a negative number.

APPENDIX B

ASCII Hexadecimal to Decimal Conversion Table

CHARACTER	HEXADECIMAL	DECIMAL
NUL	00	000
SOH	01	001
STX	02	002
ETX	03	003
EOT	04	004
END	05	005
ACK	06	006
BEL	07	007
BS	08	008
HT	09	009
LF	0A	010
VT	0B	011
FF	0C	012
CR	0D	013
SO	0E	014
SI	0F	015
DLE	10	016
DC1	11	017
DC2	12	018
DC3	13	019
DC4	14	020
NAK	15	021
SYN	16	022
ETB	17	023
CAN	18	024
EM	19	025
SUB	1A	026
ESC	1B	027
FS	1C	028
GS	1D	029
RS	1E	030
US	1F	031
SP	20	032
!	21	033
"	22	034
#	23	035
$	24	036
%	25	037
&	26	038
'	27	039
(	28	040
)	29	041
*	2A	042
+	2B	043
,	2C	044
-	2D	045
.	2E	046
/	2F	047
0	30	048
1	31	049
2	32	050
3	33	051
4	34	052
5	35	053
6	36	054
7	37	055
8	38	056
9	39	057
:	3A	058
;	3B	059
<	3C	060
=	3D	061
>	3E	062
?	3F	063
@	40	064
A	41	065
B	42	066
C	43	067
D	44	068
E	45	069
F	46	070
G	47	071
H	48	072
I	49	073
J	4A	074
K	4B	075
L	4C	076
M	4D	077
N	4E	078
O	4F	079
P	50	080
Q	51	081
R	52	082
S	53	083
T	54	084
U	55	085
V	56	086
W	57	087
X	58	088
Y	59	089
Z	5A	090
(	5B	091
\	5C	092
)	5D	093
^	5E	094
_	5F	095
`	60	096
a	61	097
b	62	098
c	63	099
d	64	100
e	65	101
f	66	102
g	67	103
h	68	104
i	69	105
j	6A	106
k	6B	107
l	6C	108
m	6D	109
n	6E	110
o	6F	111
p	70	112
q	71	113
r	72	114
s	73	115
t	74	116
u	75	117
v	76	118
w	77	119
x	78	120
y	79	121
z	7A	122
{	7B	123
|	7C	124
}	7D	125
~	7E	126
DEL	7F	127

APPENDIX C
Memory Map

ØØØØ - ØØFF	Input buffer and temporary variable storage
Ø1ØØ - 1F83	BASIC interpreter

Useful Locations

ØØ2A - ØØ2B	Contains the next available memory location after the BASIC program.
ØØ2E - ØØ2F	Contains the start of the BASIC source program.
ØØ5D - ØØ5E	Contains the address of the current arithmetic value in use during a USER call.
ØØ67 - ØØ68	Contains the pointer for USER.
Ø14E - Ø14F	This location tells BASIC where to start allocating memory for programs and variable storage. Currently this location contains the lowest possible address of 1F83. If, for example, you desire to store a 100 byte machine code program you can allocate memory from 1F83 to 2Ø83 by changing Ø14E to 2Ø83.
Ø15Ø	Contains the number of the port which BASIC will initialize to. This location currently contains Ø1, the control port.
Ø153	Contains the hex ASCII value of the line delete character. This location is normally a 18 (CTRL. X) but may be changed if desired.
Ø154	Contains the hex ASCII value that BASIC interprets as a backspace. This location is currently a Ø8 (CTRL. H). This location can be changed for terminals which do not generate an automatic cursor left with some other backspace command.
Ø155 17	Contains the character echoed to the terminal when a backspace command is entered. This character is currently a ØØ (a null). If desired this character can be changed to something else if your terminal does not generate an automatic cursor left upon sending a backspace command, such as the CT-1024 terminal.
Ø156	Contains the character which BASIC interprets as a break. Currently a Ø3 (CTRL. C).
Ø157	Contains the character that BASIC interprets as the "halt printing" command. Currently a 1B (ESC).
Ø158	Contains the bit pattern used to initialize the ACIA on MP-S type interfaces. Currently a 15 for 1 stop bit operation. A 11 should be used for 2 stop bit operation.

Below is a list of the I/O jumps in BASIC for the various ports. For each port the first is the "output character in accumulator A" jump, the second receives input and places it in accumulator A and the third is the initialization routine for a particular type port (ACIA or PIA). This I/O can be changed at the discretion of the user if desired.

```
135                          *PORT 0
136  0106 7E 1E 42  JMPTAB   JMP     POOUT
137  0109 7E 1E 36           JMP     POIN
138  010C 7E 1E 2D           JMP     POINT
139                          *PORT 1
140  010F 7E 01 5C           JMP     OUTVEC
141  0112 7E 01 67           JMP     INVEC
142  0115 7E 03 6C           JMP     FNDTYP
143                          *PORT 2
144  0118 7E 04 12           JMP     OUTACI
145  011B 7E 04 03           JMP     INACIA
146  011E 7E 03 7B           JMP     ACIINZ
147                          *PORT 3
148  0121 7E 04 12           JMP     OUTACI
149  0124 7E 04 03           JMP     INACIA
150  0127 7E 03 7B           JMP     ACIINZ
151                          *PORT 4
152  012A 7E 04 4A           JMP     OUTPIA
153  012D 7E 04 3F           JMP     INPIA
154  0130 7E 03 87           JMP     PIAINZ
155                          *PORT 5
156  0133 7E 04 4A           JMP     OUTPIA
157  0136 7E 04 3F           JMP     INPIA
158  0139 7E 03 87           JMP     PIAINZ
159                          *PORT 6
160  013C 7E 04 4A           JMP     OUTPIA
161  013F 7E 04 3F           JMP     INPIA
162  0142 7E 03 87           JMP     PIAINZ
163                          *PORT 7
164  0145 7E 04 4A           JMP     OUTPIA
165  0148 7E 04 3F           JMP     INPIA
166  014B 7E 03 87           JMP     PIAINZ
167  014E 1F 83      FILE    FDB     END
168  0150 01         INZPOR  FCB     1
169  0151 03 37      VEREND  FDB     COMEND
170  0153 18         DELCHR  FCB     $18
171  0154 08         BACCHR  FCB     $08
172  0155 00         ARRCHR  FCB     $00
173  0156 03         BRKCHR  FCB     $03
174  0157 1B         STPCHR  FCB     $1B
175  0158 15         STOPBT  FCB     $15          STOP BITS

177  0159 7E 1E 6F  OFF      JMP     POOFF

179  015C 7D 00 1F  OUTVEC   TST     TYPE
180  015F 26 03              BNE     *+5
181  0161 7E 04 12           JMP     OUTACI
182  0164 7E E1 D1           JMP     OUTEEE
183  0167 7D 00 1F  INVEC    TST     TYPE
184  016A 26 03              BNE     *+5
185  016C 7E 04 03           JMP     INACIA
186  016F 7E E1 AC           JMP     INEEE
```

APPENDIX D
Notes for Speeding up BASIC

1. Subscripted variables take considerable time; whenever possible use non-subscripted variables.
2. Transcendental functions (SIN, COS, TAN, ATAN, EXP, LOG) are slow because of the number of calculations involved, so use them only when necessary. Also, the ↑ operator uses both the LOG and the EXP functions, so use the format A*A to square a number.
3. BASIC searches for functions and subroutines in the source file, so place often called routines at the start of the program.
4. BASIC searches the symbol table for a referenced variable, and variables are inserted into the symbol table as they are referenced; therefore, reference a frequently called variable early in the program so that it comes early in the symbol table.
5. Numeric constants are converted each time they are encountered, so if you use a constant often, assign it to a variable and use the variable instead.

APPENDIX E
Notes on Memory Usage in BASIC

1. REM statements use space, so use them sparingly.

2.a. Each non-subscripted numeric variable takes 8 bytes.

b. Each non-subscripted string variable takes 34 bytes.

c. Each numeric array takes 6 bytes plus 6 bytes for each element.

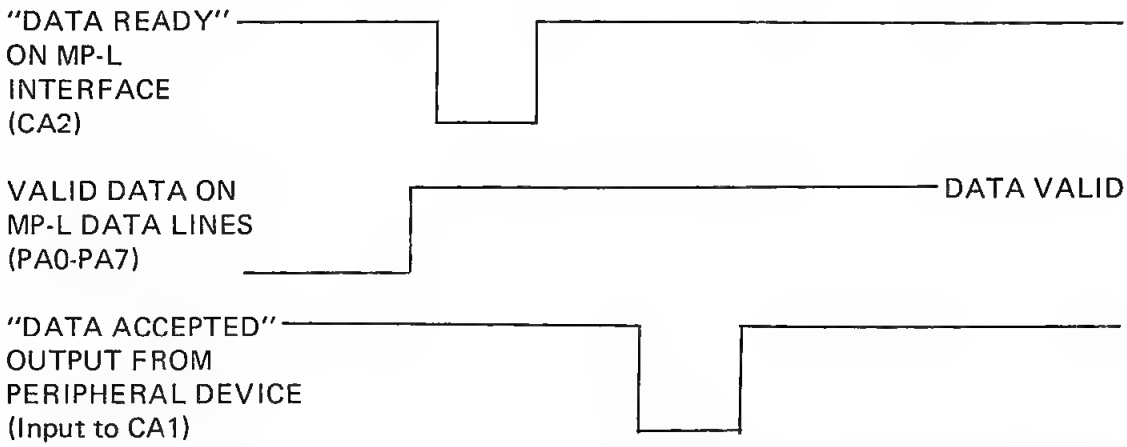
d. Each string array takes 6 bytes plus 32 bytes for each element.

3. An implicitly dimensioned variable creates 10 elements (or 10 by 10). If you do not intend to use all 10 elements, save memory by explicitly dimensioning the variable.
4. Each BASIC line takes 2 bytes for the line number, 2 bytes for the encoded key word, 1 byte for the end of line terminator, 1 byte for the line length, plus one byte for each character following the key word. Memory can be saved by using as few spaces as possible.
5. BASIC reserves the uppermost 256 bytes of memory for stack and buffer use.

APPENDIX F

Parallel Interface Handshaking

The parallel interface drivers are written for a conventional handshaking scheme used by many printer manufacturers and is the same as that on a SWTPC PR-40 printer. Handshaking timing is set up as follows:



† peripheral should generate this pulse when it has accepted the previous data and is ready for more.

APPENDIX G

PIA Strobing

Use of the Control Interface on Port #Ø for Read/Punch—On/Off Decoding

BASIC software contains subroutines to send out pulses to unused pins of the PIA integrated circuit on the MP-C serial control interface that can be used for automatic reader/punch controls. These pulses can be used if you are using a SWTPC AC-30 cassette interface and a termimal in which access to the control command decoding is denied.

If you intend to use the read/punch control logic output on the MP-C control interface board, make the following connections from the indicated pins of IC1 on the MP-C control interface board to the specified pins of a twelve pin male connector shell. The connector pinning shown below is correct for a SWTPC AC-30 cassette interface and will need modification for other units. Be sure to make the wires long enough to reach your AC-30 where the connector will be plugged. If you have access to your terminal's 16X baud rate clock, the terminal's clock bus should be broken and the 16X clock IN and OUT lines brought out to the same connector.

MP-C IC1 pin 7 (read on)	12 pin male shell female pin 1
MP-C IC1 pin 4 (punch on)	12 pin male shell female pin 2
MP-C IC1 pin 6 (read off)	12 pin male shell female pin 3
MP-C IC1 pin 5 (punch off)	12 pin male shell female pin 4
Terminal's 16X clock OUT	12 pin male shell female pin 5
Terminal's 16X clock IN	12 pin male shell female pin 6
MP-C ground	12 pin male shell female pin 12

These signals are low going pulses and are about 15 microseconds wide. They are not buffered and should drive a maximum of only one standard TTL load.

PIA strobing will work only on a MP-C type interface on port #Ø and only on systems containing a SWTBUG® monitor.

Instruction Set Summary

Commands	Statements	Functions
APPEND	DATA	ABS(X)
CONT	READ	CHR$(X)
DIGITS=&	RESTORE	COS(X)
LINE= &	FOR /exp/ to /exp/	DEF FN(X)
LIST &	NEXT	EXP(X)
LOAD	GOSUB	INT(X)
NEW	DIM *	LEFT$(X$,N)
PATCH	END	LEN(X$)
PORT= &	GOTO *	MID$(X$,S.Y)
RUN	IF /rel. exp./ THEN /line/	PEEK(X)
SAVE	INPUT *	POKE (I,J)
STRING=&	ON /exp/GOT/line(s)/	POS
TRACE ON &	ON /exp/ GOSUB/line(s)/	RIGHT(X$,N)
TRACE OFF &	PRINT *	RND(X)
	REM	SGN(X)
	RETURN	SIN(X)
	STOP	SQR(X)
		STR$(X)
		TAB(X)
		TAN(X)
		TAN(X)
		USER(X)
		VAL(X$)

* Denotes statements that may be used in the DIRECT mode
& Denotes commands that may be used as program statement

MATH OPERATORS		RELATIONAL OPERATORS	
↑	Exponentiate	=	Equal (numeric and string)
—	(unary) Negate	<>	Not Equal (numeric and string)
*	Multiplication	<	Less Than
/	Division	>	Greater Than
+	Addition, string concatenation	<=	Less Than or Equal
—	Subtraction	>=	Greater Than or Equal

LINE NUMBERS —May be from 1 thru 9999
VARIABLES —May be single character alphabetic or single character alphabetic followed by one integer 0 thru 9 or $
BACKSPACE —Is a Control H
LINE CANCEL —Is a Control X (CANCEL)
PROGRAM ABORT—Typing a Control C should bring BASIC back to the READY mode regardless of what the BASIC program is doing (except USER programs).
LINES —Each line may contain multiple statements. Each statement is separated from the other with a : .

INDEX

0000

1F80

1F8AF [illegible]

1F70 CF 0B C9 00 00 0D

0100

1F83

1F70-

1FC0-01 8A 0B DF [illegible] [illegible] 1A [illegible] [illegible] [illegible]

1F80 7F 8D 2E [illegible] 21

[illegible]

[illegible]

[illegible]

1E00 B6 0004 47 24
1F00 25 B7